UNIVERSIDAD DE MÁLAGA

Modelado y Diseño de Software

Examen Febrero 2021

Nombre: _____

PARTE 1 (15 minutos)

A. Defina los siguientes conceptos [0,-2]

- Ingeniería de software
- Modelo
- Diseño de software
- Patrón de diseño

B. En UML, ¿qué significa que uno de los extremos de una asociación tenga un círculo negro (“ ——● ”)? [0.25]

C. Indique las principales semejanzas y diferencias entre los patrones de diseño “Decorador” y “Compuesto” [0.25]

PARTE 2 (2 horas 15 minutos)

P1. Los teatros

En una ciudad hay un conjunto de teatros, que a lo largo de la temporada representan diversas obras. Cada obra tiene un título, un autor y un reparto, que no es sino el conjunto de personajes que aparecen en ella. Cada obra se representa varias veces a lo largo de una temporada, en días concretos y en sesiones de tarde y noche. Los actores son personas que interpretan a los personajes de una obra en una representación concreta. En una misma representación, un personaje solo puede ser interpretado por un actor, aunque un mismo actor puede interpretar a varios personajes. Un actor puede actuar en representaciones distintas el mismo día (de la misma o de obras distintas), siempre que no coincidan en la misma sesión (tarde o noche). Se pide:

(a) Desarrollar el modelo conceptual de la estructura de dicho sistema utilizando la herramienta USE, incluyendo las restricciones apropiadas que garanticen la coherencia de los datos que maneja el sistema.

(b) Especificar en OCL las siguientes operaciones de consulta (*queries*) en la clase que representa a un teatro:

- Una operación “obras()” para conocer el conjunto de obras que se han representado en el teatro en cualquier temporada.
- Una operación “obrasTemp()”que permita conocer el conjunto de obras que se han representado en el teatro en una temporada concreta (que se le ha de pasar como parámetro).

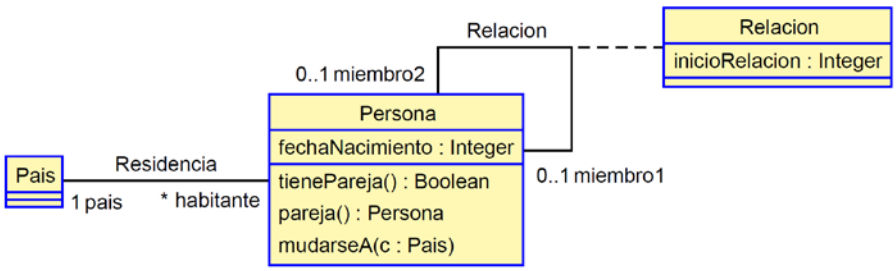
(c) Especificar en OCL los dos siguientes invariantes adicionales:

- Un actor puede actuar en varios teatros cada temporada, aunque solamente en una única obra de cada uno de ellos.
- Varios teatros pueden representar la misma obra en una temporada, pero no pueden hacerlo en los mismos días.

(d) Supongamos además que cada teatro obtiene unos beneficios por cada representación de una obra, que provienen de las tarifas que cobra a los asistentes. Los asistentes a una representación pagan una tarifa que se calcula multiplicando por 10 Euros el número de personajes que aparecen en el reparto de la obra. Aquellos asistentes que son socios del teatro pagan la mitad de la tarifa, pero abonan al teatro 1000 Euros por temporada. Se pide extender el modelo del sistema para incluir esta nueva información y especificar en OCL una operación que calcule los ingresos de un teatro en una temporada dada.

P2. Parejas

Supongamos el modelo conceptual mostrado a continuación, que representa relaciones de pareja que pueden tener algunas personas. Cada persona vive en un país, y puede tener a lo sumo una relación de pareja en un momento dado.



Las fechas se indican en años. Supongamos además las siguientes restricciones de integridad sobre el modelo:

```
context Persona inv FechaNacimientoOK:
        self.fechaNacimiento>=0
context Relacion inv MiembrosDistintos:
        self.miembro1<>self.miembro2
context Persona inv SoloUnRol:
        self.miembro1<>null implies self.miembro2=null and
        self.miembro2<>null implies self.miembro1=null
context Relacion inv InicioOK:
        self.inicioRelacion>=self.miembro1.fechaNacimiento+18 and
        self.inicioRelacion>=self.miembro2.fechaNacimiento+18
context Relacion inv MismoPais:
        self.miembro1.pais = self.miembro2.pais
```

La primera restricción pide que las fechas de nacimiento sean positivas. La segunda impide que una persona sea pareja de sí misma. La tercera indica que si existe una relación (p1,p2) en el sistema, no puede existir la relación (p2,p1), al considerarse de forma lógica que son la misma pareja. La cuarta indica que las personas solo pueden emparejarse si tienen más de 18 años. Finalmente, la quinta restricción indica que en este sistema las parejas deben residir en el mismo país.

Se pide:

a) Especificar con pre y post-condiciones la operación de la clase Persona "mudarseA()", que cambia el lugar de residencia de la persona. Dichas pre y post-condiciones deben respetar y garantizar el cumplimiento de los invariantes del modelo conceptual. Especificar también las operaciones "tienePareja()" que devuelve si la persona tiene pareja o no, y "pareja()" que devuelve la pareja actual de una persona (o null si no tiene relación en este momento).
b) Convertir el modelo conceptual indicado arriba en un modelo de diseño, utilizando la herramienta MagicDraw, que permita la implementación, en un lenguaje orientado a objetos como Java, de las entidades y asociaciones correspondientes. Incluir en el modelo de diseño explícitamente los correspondientes constructores, getters, setters y demás operaciones que se consideren adecuadas, indicando su visibilidad y argumentos.
c) Desarrolle una implementación en Java correspondiente a dicho modelo de diseño, que incorpore el código de andamiaje para crear objetos y relaciones entre ellos garantizando en todo momento la coherencia de los objetos y de las relaciones de la implementación, y las restricciones e invariantes del modelo.
d) Implementar también las operaciones "mudarseA()", "tieneParejа()" y "pareja()" de acuerdo al modelo de diseño realizado, y respetando la especificación definida en el apartado (a)

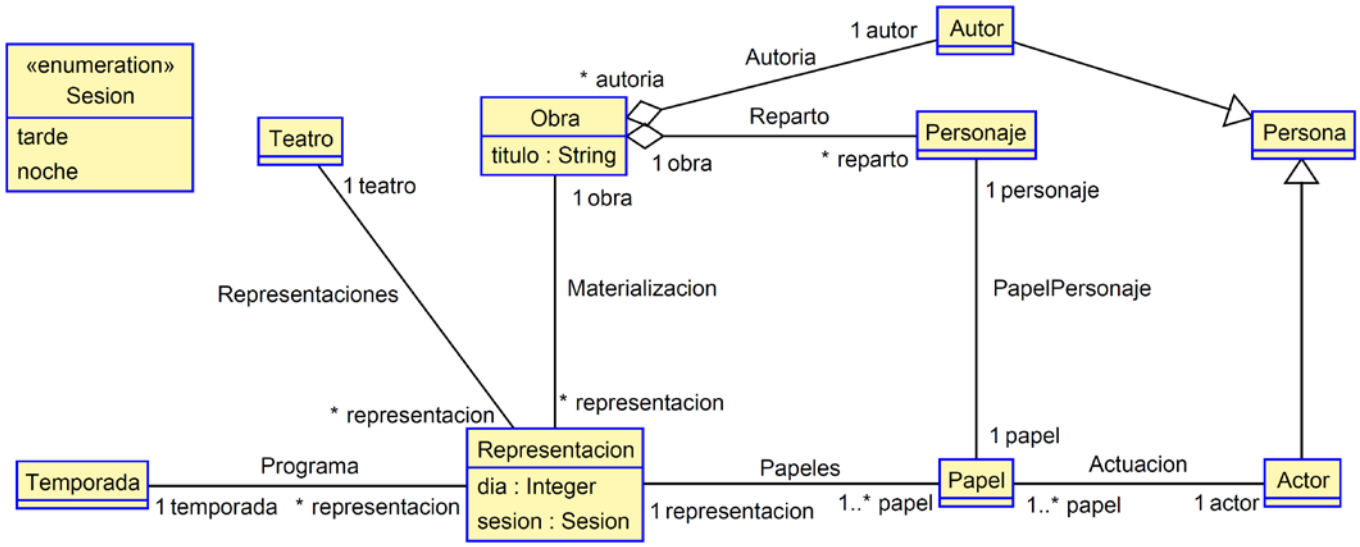
- **Puntuaciones**: P1 (2.5+0.5+0.5+1.5) P2 (0.75+0.5+2.5+0.75). Total: 9.5
- **Examen reducido**: Problemas P1(a), P1(b), P1(c), P2(b) y P2(c). Total: 6.5 (nota a multiplicar por 9.5/6.5 si se supera el 3.5)
- **Entrega**: La entrega se hará a través del campus virtual, en un solo archivo **en formato PDF**, que contendrá una memoria explicativa del examen y en donde se describirán las soluciones propuestas para cada problema y se incluirán todas las imágenes con los diagramas UML, así como todos los códigos en Java y USE desarrollados.

Soluciones

Problema 1. Apartado (a)

Desarrollar el modelo conceptual de la estructura de dicho sistema utilizando la herramienta USE, incluyendo las restricciones apropiadas que garanticen la coherencia de los datos que maneja el sistema.

Un modelo que representa ese sistema es el siguiente.



Es importante observar la necesidad de utilizar una clase “Papel” que sirva como relación ternaria entre Actor, Personaje y Representación. El código en USE correspondiente a ese diagrama es el siguiente:

```
model Teatro
enum Sesion {tarde, noche}

class Teatro
end

class Obra
attributes
   titulo:String
constraints
   inv tituloOK: titulo<>null and titulo<>''
end

class Persona
end

class Autor < Persona
end

aggregation Autoria between
   Obra [*] role autoria
   Autor [1] role autor
end

aggregation Reparto between
   Obra [1] role obra
   Personaje [*] role reparto
end

class Personaje
end
```

```
class Papel
end

association PapelPersonaje between
   Papel [1] role papel
   Personaje [1] role personaje
end

association Actuacion between
   Papel [1..*] role papel
   Actor [1] role actor
end

association Papeles between
   Papel [1..*] role papel
   Representacion [1] role representacion
end

class Actor < Persona
end

class Temporada
end

association Representaciones between
   Teatro [1] role teatro
   Representacion [*] role representacion
end

association Programa between
   Temporada [1] role temporada
   Representacion [*] role representacion
end

class Representacion
attributes
   dia:Integer
   sesion:Sesion
constraints
   inv diaOK: dia>=0
end

association Materializacion between
   Obra [1] role obra
   Representacion [*] role representacion
end

constraints
-- Un actor no puede actuar a la vez en dos sesiones
context Actor inv NoActuacionesSimultaneas:
   self.papel.representacion->forAll(r1,r2|r1<>r2 implies (r1.dia=r2.dia implies r1.sesion<>r2.sesion))
```

Apartado (b) operationes en la clase Teatro:

Especificar en OCL las siguientes operaciones de consulta (queries) en la clase que representa a un teatro:

- *Una operación "obras()" que permita conocer el conjunto de obras que se han representado en el teatro en cualquier temporada.*
- *Una operación "obrasTemp()"que permita conocer el conjunto de obras que se han representado en el teatro en una temporada concreta (que se le ha de pasar como parámetro).*

obras():Set(Obra) = self.representacion->collect(obra)->asSet()

obrasTemp(t:Temporada):Set(Obra) = self.representacion->select(temporada=t)->collect(obra)->asSet()

Apartado (c) Restricciones que nos piden

Especificar en OCL los dos siguientes invariantes adicionales:

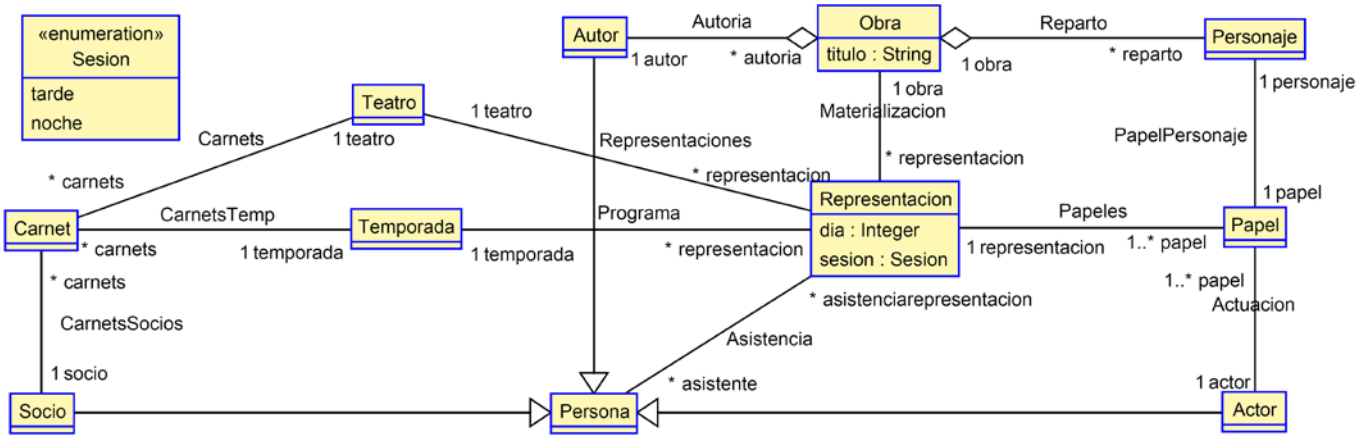
- *Un actor puede actuar en varios teatros cada temporada, aunque solamente en una única obra de cada uno de ellos.*
- *Varios teatros pueden representar la misma obra en una temporada, pero no pueden hacerlo en los mismos días.*

```
context Actor inv NoMasDeUnaObraPorTeatroYTemporada:
Temporada.allInstances->forAll(temp|
  Teatro.allInstances->forAll(tea|
    self.papel.representacion->select(r|r.temporada=temp and r.teatro=tea)->collect(obra)->size()<=1
  )
)

context Representacion inv DiasDiferentesMismaObra:
  Representacion.allInstances->forAll(r1,r2| r1<>r2 implies
     ((r1.temporada=r2.temporada and r1.obra=r2.obra and r1.teatro<>r2.teatro) implies (r1.dia <> r2.dia))
  )
```

Apartado (d)

Supongamos además que cada teatro obtiene unos beneficios por cada representación de una obra, que provienen de las tarifas que cobra a los asistentes. Los asistentes a una representación pagan una tarifa que se calcula multiplicando por 10 Euros el número de personajes que aparecen en el reparto de la obra. Aquellos asistentes que son socios del teatro pagan la mitad de la tarifa, pero abonan al teatro 1000 Euros por temporada. Se pide extender el modelo del sistema para incluir esta nueva información y especificar en OCL una operación que calcule los ingresos de un teatro en una temporada dada.

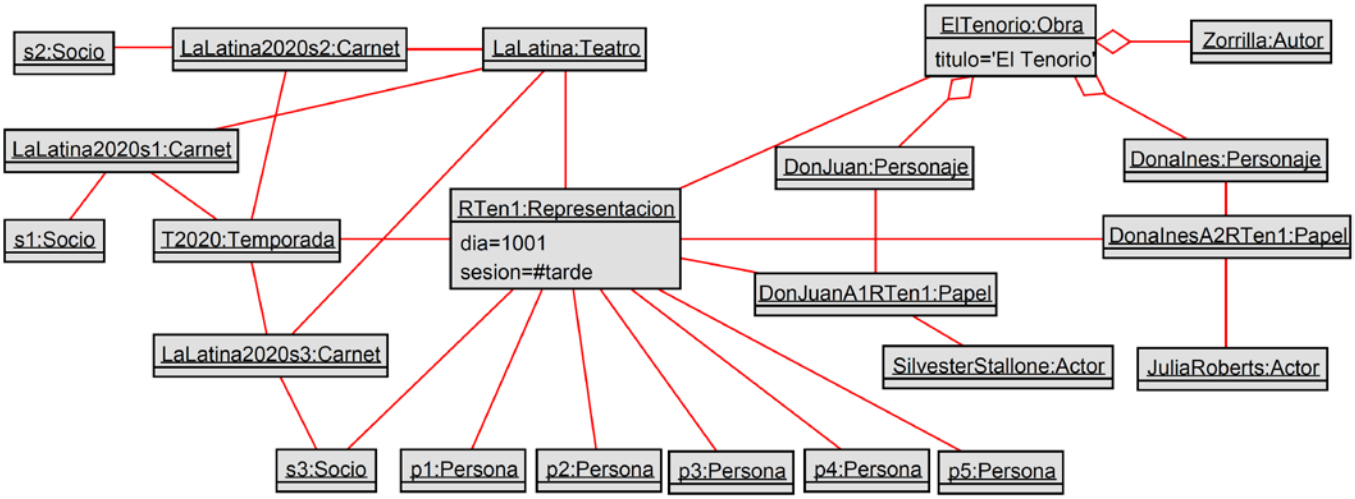
Lo primero es extender el modelo conceptual con la clase "Socio", que hereda de "Persona", la asociación "Asistencia" entre "Persona" y "Representacion", y finalmente la clase "Carnet" que representa una relación ternaria entre las clases "Socio", "Teatro" y "Temporada". La clase Carnet es necesaria porque es preciso representar un objeto por cada socio, teatro y temporada.



A partir de ahí, lo único que queda es especificar la operación que nos piden:

```
recaudacion(temp:Temporada):Integer =
-- primero los socios de esa temporada
1000 * self.carnets->select(s|s.temporada=temp)->size()
+ -- Y luego la asistencia a cada representacion de esa temporada
self.representacion->select(r|r.temporada=temp)->collect(asistente)->
  iterate(a;income:Integer=0 | income +
      if (a.oclIsTypeOf(Socio) and
         a.oclAsType(Socio).carnets->select(s|s.teatro=self and s.temporada=temp)->notEmpty())
      then 5
      else 10
      endif)
```

Finalmente, los siguientes comandos soil permiten crear un modelo de objetos que sirva de pruebas básicas para los diferentes modelos.




```
reset
!new Teatro('LaLatina')

!new Temporada('T2020')

!new Autor('Zorrilla')

!new Obra('ElTenorio')
!ElTenorio.titulo:='El Tenorio'
!insert (ElTenorio,Zorrilla) into Autoria
!new Personaje('DonJuan')
!insert(ElTenorio,DonJuan) into Reparto
!new Personaje('DonaInes')
!insert(ElTenorio,DonaInes) into Reparto

!new Representacion('RTen1')
!RTen1.dia:=1001
!RTen1.sesion:=#tarde
!insert (ElTenorio,RTen1) into Materializacion
!insert (LaLatina,RTen1) into Representaciones
!insert (T2020,RTen1) into Programa
!new Actor('SilvesterStallone')
!new Actor('JuliaRoberts')
!new Papel('DonJuanA1RTen1')
!new Papel('DonaInesA2RTen1')
```

```
!insert (DonJuanA1RTen1,DonJuan) into PapelPersonaje
!insert (DonJuanA1RTen1,RTen1) into Papeles
!insert (DonJuanA1RTen1,SilvesterStallone) into Actuacion
!insert (DonaInesA2RTen1,DonaInes) into PapelPersonaje
!insert (DonaInesA2RTen1,RTen1) into Papeles
!insert (DonaInesA2RTen1,JuliaRoberts) into Actuacion

-- SOCIOS
!new Carnet('LaLatina2020s1')
!new Carnet('LaLatina2020s2')
!new Carnet('LaLatina2020s3')
!new Socio('s1')
!new Socio('s2')
!new Socio('s3')
!insert (T2020,LaLatina2020s1) into CarnetsTemp
!insert (LaLatina,LaLatina2020s1) into Carnets
!insert (s1,LaLatina2020s1) into CarnetsSocios
!insert (T2020,LaLatina2020s2) into CarnetsTemp
!insert (LaLatina,LaLatina2020s2) into Carnets
!insert (s2,LaLatina2020s2) into CarnetsSocios
!insert (T2020,LaLatina2020s3) into CarnetsTemp
!insert (LaLatina,LaLatina2020s3) into Carnets
!insert (s3,LaLatina2020s3) into CarnetsSocios

-- ASISTENCIA
!new Persona('p1')
!new Persona('p2')
!new Persona('p3')
!new Persona('p4')
!new Persona('p5')
!insert (RTen1,p1) into Asistencia
!insert (RTen1,p2) into Asistencia
!insert (RTen1,p3) into Asistencia
!insert (RTen1,p4) into Asistencia
!insert (RTen1,p5) into Asistencia

!insert (RTen1,s3) into Asistencia

check
?LaLatina.obras()                -- Result: -> Set{ElTenorio} : Set(Obra)
?LaLatina.obrasTemp(T2020)       -- Result: -> Set{ElTenorio} : Set(Obra)
?LaLatina.recaudacion(T2020)     -- Result: -> 3055 : Integer
```

Segundo problema

Apartado (a)

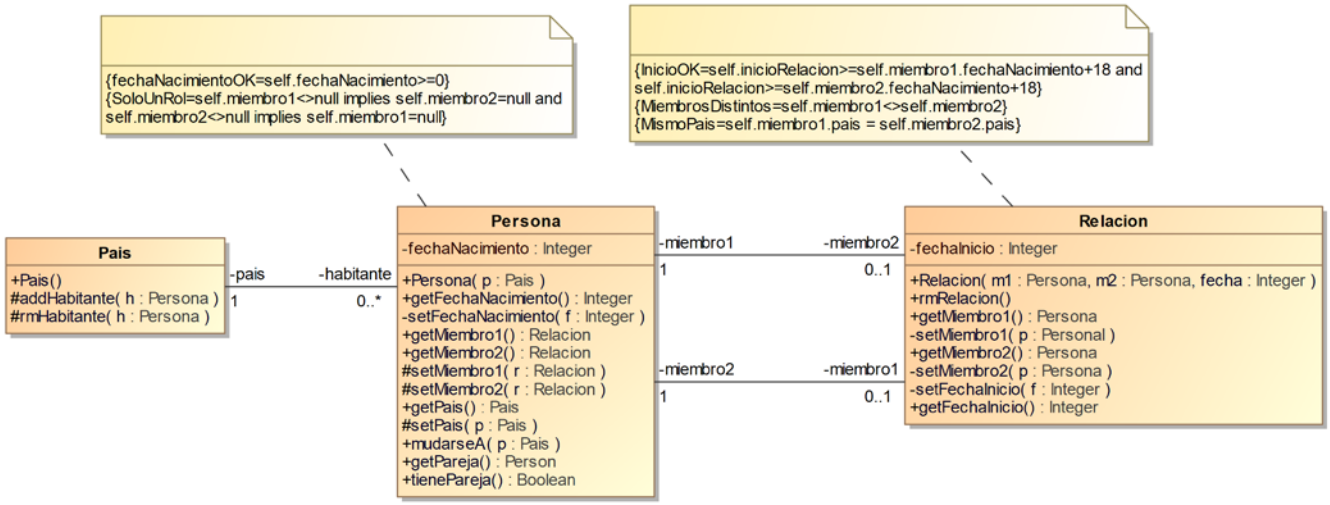
Especificar con pre y post-condiciones la operación de la clase Persona "mudarseA()", que cambia el lugar de residencia de la persona. Dichas pre y post-condiciones deben respetar y garantizar el cumplimiento de los invariantes del modelo conceptual. Especificar también las operaciones "tienePareja()" que devuelve si la persona tiene pareja o no, y "pareja()" que devuelve la pareja actual de una persona (o null si no tiene relación en este momento).

El modelo completo del sistema en USE, incluyendo las especificaciones que nos piden, es el siguiente.

```
model Parejas

class Persona
attributes
   fechaNacimiento:Integer
operations
  tienePareja():Boolean = self.miembro1<>null or self.miembro2<>null
  pareja():Persona = if self.miembro1<>null then self.miembro1
                          else  if self.miembro2<>null then self.miembro2
                                else null
                                endif
                          endif
  mudarseA(c:Pais)
   begin
      insert(self,c) into Residencia
   end
   pre mismoPaisPareja:
      self.miembro1<>null implies self.miembro1.pais=c and
      self.miembro2<>null implies self.miembro2.pais=c
   post ActualizaPais: self.pais=c and c.habitante->includes(self)
end

class Pais
end

association Residencia between
   Persona [*] role habitante
   Pais [1] role pais
end

associationclass Relacion between
   Persona [0..1] role miembro1
   Persona [0..1] role miembro2
attributes
   inicioRelacion:Integer
end

constraints
context Relacion inv MiembrosDistintos: self.miembro1<>self.miembro2
context Persona inv FechaNacimientoOK: self.fechaNacimiento>=0
context Persona inv SoloUnRol: self.miembro1<>null implies self.miembro2=null and
                        self.miembro2<>null implies self.miembro1=null
context Relacion inv InicioOK: self.inicioRelacion>=self.miembro1.fechaNacimiento+18 and
                       self.inicioRelacion>=self.miembro2.fechaNacimiento+18
context Relacion inv MismoPais: self.miembro1.pais = self.miembro2.pais
```

Apartado (b)

Convertir el modelo conceptual indicado arriba en un modelo de diseño, utilizando la herramienta MagicDraw, que permita la implementación, en un lenguaje orientado a objetos como Java, de las entidades y asociaciones correspondientes. Incluir en el modelo de diseño explícitamente los correspondientes constructores, getters, setters y demás operaciones que se consideren adecuadas, indicando su visibilidad y argumentos.



Apartados c) y d)

c) Desarrolle una implementación en Java correspondiente a dicho modelo de diseño, que incorpore el código de andamiaje para crear objetos y relaciones entre ellos garantizando en todo momento la coherencia de los objetos y de las relaciones de la implementación, y las restricciones e invariantes del modelo.

d) Implementar también las operaciones "mudarseA()", "tienePareja()" y "pareja()" de acuerdo al modelo de diseño realizado, y respetando la especificación definida en el apartado (a)

```
import java.util.*;
//CLASS PAIS
public class Pais {
	// uses default constructor
	List<Persona> habitante = new LinkedList<Persona>();

	void addHabitante(Persona p){
		assert(p!=null);
		if (p.getPais()!=null)
			p.getPais().rmHabitante(p);  // removes it from its current country
		habitante.add(p);                    // adds it to its new country
	}
	void rmHabitante(Persona p) {
		assert(p!=null);
		habitante.remove(p);
	}
	public Enumeration<Persona> getHabitante(){
		return java.util.Collections.enumeration(habitante);
	}
}
//CLASS PERSONA
public class Persona {
	private Pais pais;
	private int fechaNacimiento;
	private Relacion miembro1;
	private Relacion miembro2;
	//CONSTRUCTOR
	public Persona(int fecha, Pais p) {
		assert(fecha>=0); assert(p != null);
		setFechaNacimiento(fecha);
		this.setPais(p);      // updates its state
	}
```

```
        //SETTERS & GETTERS
        void setPais(Pais p) {
                assert(!this.tienePareja() || p==this.getPais());
                if (this.pais!=null) this.pais.rmHabitante(this); //remove from current country
                p.addHabitante(this); //add to new country
                pais=p; // updates its state
        }
        public Pais getPais() {
                return pais;
        }
        private void setFechaNacimiento(int f) {
                assert(f>=0);
                fechaNacimiento=f;
        }
        public int getFechaNacimiento() {
                return fechaNacimiento;
        }
        void setMiembro1(Relacion r) {
                assert( r==null || this.getMiembro2()==null );
                miembro1=r;
        }
        Relacion getMiembro1() {
                return miembro1;
        }
        void setMiembro2(Relacion r) {
                assert( r==null || this.getMiembro1()==null );
                miembro2=r;
        }
        Relacion getMiembro2() {
                return miembro2;
        }
        public void mudarseA(Pais c) {
                this.setPais(c);
        }
        public boolean tienePareja() {
                return (this.getMiembro1()!=null || this.getMiembro2()!=null);
        }
        public Persona pareja() {
                if (this.getMiembro1()!=null) return this.getMiembro1().getMiembro2();
                if (this.getMiembro2()!=null) return this.getMiembro2().getMiembro1();
                return null;
        }
}

//CLASS RELACION
public class Relacion {

        private Persona miembro1;
        private Persona miembro2;
        private int fechaInicio;

        public Persona getMiembro1() {
                return miembro1;
        }
        public Persona getMiembro2() {
                return miembro2;
        }
        private void setMiembro1(Persona m) { //no external setter
                this.miembro1 = m;
        }
        private void setMiembro2(Persona m) { //no external setter
                this.miembro2 = m;
        }
        private void setFechaInicio(int f) { //no external setter
                this.fechaInicio=f;
        }

        //CONSTRUCTOR
```

```
	public Relacion(Persona m1, Persona m2, int fecha){
		//PRECONDITIONS
		assert(m1!=m2);
		assert(fecha >= m1.getFechaNacimiento() + 18);
		assert(fecha >= m2.getFechaNacimiento() + 18);
		assert(m1.getPais()==m2.getPais());
		//Main body
		this.setMiembro1(m1);
		this.setMiembro2(m2);
		this.setFechaInicio(fecha);
		m1.setMiembro1(this); // To ensure consistency
		m2.setMiembro2(this); // To ensure consistency
	}
	//BEFORE DELETING THE OBJECT
	public void rmRelacion() {
		miembro1.setMiembro1(null);
		miembro2.setMiembro2(null);
		this.setMiembro1(null);
		this.setMiembro2(null);
	}
}

//CLASS TEST – SOME VERY BASIC TESTS TO CHECK FOR INITIAL CORRECTNESS
import java.util.*;
public class Test {
	public static void main(String[] args) {
		Pais c1 = new Pais();
		Pais c2 = new Pais();
		Persona p1 = new Persona(1962,c1);
		Persona p2 = new Persona(1963,c1);
		Persona p3 = new Persona(1964,c2);

		assert(p1.getPais()==c1);
		assert(p2.getPais()==c1);
		assert(p3.getPais()==c2);
		Enumeration<Persona> t1 = c1.getHabitante();
		Enumeration<Persona> t2 = c2.getHabitante();
		assert(t1.nextElement()==p1);
		assert(t1.nextElement()==p2);
		assert(!t1.hasMoreElements());
		assert(t2.nextElement()==p3);
		assert(!t2.hasMoreElements());

		assert(!p1.tienePareja());
		assert(!p2.tienePareja());
		assert(!p3.tienePareja());

		Relacion r1 = new Relacion(p1,p2,2000);
		assert(p1.tienePareja() && p2.tienePareja() && p1.pareja()==p2 && p2.pareja()==p1);
		assert(!p3.tienePareja());

		r1.rmRelacion();
		assert(!p1.tienePareja());
		assert(!p2.tienePareja());
		assert(!p3.tienePareja());

		p2.mudarseA(c2);
		assert(p1.getPais()==c1);
		assert(p2.getPais()==c2);
		assert(p3.getPais()==c2);
		t1 = c1.getHabitante();
		t2 = c2.getHabitante();
		assert(t1.nextElement()==p1);
		assert(!t1.hasMoreElements());
		assert(t2.nextElement()==p3);
		assert(t2.nextElement()==p2);
		assert(!t2.hasMoreElements());
```

```
        Relacion r2 = new Relacion(p2,p3,2000);
        assert(p2.tienePareja() && p3.tienePareja() && p3.pareja()==p2 && p2.pareja()==p3);
        assert(!p1.tienePareja());
        r2.rmRelacion();
        assert(!p1.tienePareja());
        assert(!p2.tienePareja());
        assert(!p3.tienePareja());

        p1.mudarseA(c2);
        assert(p1.getPais()==c2);
        assert(p2.getPais()==c2);
        assert(p3.getPais()==c2);
        t1 = c1.getHabitante();
        t2 = c2.getHabitante();
        assert(!t1.hasMoreElements());
        assert(t2.nextElement()==p3);
        assert(t2.nextElement()==p2);
        assert(t2.nextElement()==p1);
        assert(!t2.hasMoreElements());

        Relacion r3 = new Relacion(p1,p3,2000);
        assert(p1.tienePareja() && p3.tienePareja() && p1.pareja()==p3 && p3.pareja()==p1);
        assert(!p2.tienePareja());

        r3.rmRelacion();
        assert(!p1.tienePareja());
        assert(!p2.tienePareja());
        assert(!p3.tienePareja());

        p3.mudarseA(c1);
        assert(p1.getPais()==c2);
        assert(p2.getPais()==c2);
        assert(p3.getPais()==c1);
        t1 = c1.getHabitante();
        t2 = c2.getHabitante();
        assert(t1.nextElement()==p3);
        assert(!t1.hasMoreElements());
        assert(t2.nextElement()==p2);
        assert(t2.nextElement()==p1);
        assert(!t2.hasMoreElements());

        //r3 = new Relacion(p1,p2,1970); raises exception
        //r3 = new Relacion(p1,p3,2000); raises exception
    }
}
```
